Неделя 7. Лаба 4. Git и GitHub: репозиторий сервиса и работа через PR

Часть 1 из 2: разделы «Подготовка: Git и GitHub» — «6. Конфликт (один раз, в репозитории A)».

Цель: репозиторий с защищённым `main`, задачи на доске, работа через ветки и PR с ревью кода, разрешение конфликта, первый релиз. **Время:** 90 мин. **Теория:** лекция 4. **Инструменты:** Git, GitHub, Visual Studio 2026 (Git Changes), PowerShell 7, браузер.

Формат: пары, два этапа примерно по 45 минут.

- **Этап 1 — репозиторий A.** A — автор, B — ревьюер. Выполняете пп. 1–8.
- **Этап 2 — репозиторий B.** Роли меняются, повторяете пп. 1–4 и 7–8.
- **Конфликт (п. 6)** делаете один раз, в репозитории A, вдвоём.

Подготовка: Git и GitHub

С этой лабы код живёт в Git и на GitHub.

Что	Зачем
Git	Хранит историю изменений, ветки; отправляет код на GitHub
Аккаунт на github.com	Репозиторий, Pull Request, ревью; в лабе 5 — CI/CD и реестр образов

```
winget install -e --id Git.Git          # или установщик с git-scm.com; затем откройте терминал заново
git --version
git config --global user.name "Имя Фамилия"
git config --global user.email "you@example.com"   # тот же email, что в аккаунте GitHub
git config --global init.defaultBranch main
```

- Зарегистрируйтесь на github.com, если аккаунта нет, и сообщите ник преподавателю.
- **Рекомендуем:** оформите GitHub Student Developer Pack — он даёт бесплатный GitHub Pro, а с ним защиту веток в приватных репозиториях (п. 1).
- Вход: при первом `git push` Git Credential Manager откроет браузер. Пароли и токены в терминал вводить не нужно.

Visual Studio тоже работает с Git: окно **Git Changes** (*Вид → Git Changes*) показывает изменения, коммиты и ветки. В лабе команды даны для терминала, но любую из них можно выполнить и там.

Начало:

```
cd C:\dev\demo-api
$K = "C:\dev\kit"              # путь к kit из лабы 3
Test-Path $K\.github           # True
```

1. Приватный или публичный репозиторий

У вас	Репозиторий	Что включить
GitHub Pro (Student Developer Pack). Проверка: *Settings → Billing and plans* → план **Pro**	**Private**	—
Нет Pro	**Public**	*Settings → Actions → General → Fork pull request workflows from outside collaborators* → **Require approval for all outside collaborators**

Зачем: правила защиты веток на приватных репозиториях есть только в Pro. В публичном репозитории чужой PR мог бы запустить свой код на вашем ПК через self-hosted runner (лаба 5); настройка требует вашего одобрения таких запусков.

2. Репозиторий

A: на GitHub *New repository* → `demo-api`, видимость по п. 1, без README.

```
git init
git add -A
git status                  # .env, bin/, obj/, .vs/ в списке быть НЕ должно
git commit -m "chore: task service with Dockerfile and Compose"
git branch -M main
git remote add origin https://github.com/<you>/demo-api.git
git push -u origin main
```

Зачем проверять `git status`: один случайно закоммиченный `.env` — это утечка пароля, которую уже не удалить из истории.

A: *Settings* → *Collaborators* → *Add people* → партнёр и преподаватель.

B: примите приглашение (письмо или страница `https://github.com/<A>/demo-api/invitations`) и склонируйте репозиторий партнёра в отдельную папку:

```
cd C:\dev
git clone https://github.com/<A>/demo-api.git demo-api-<A>
```

3. Защита `main`

Settings → *Rules* → *Rulesets* → *New ruleset* → *New branch ruleset*:

- **Name:** `main`, **Enforcement:** Active
- **Target branches:** Include default branch
- ✅ Restrict deletions
- ✅ Require a pull request before merging → Required approvals: **1**
- ✅ Block force pushes

Зачем: в `main` попадает только код, который видел второй человек. В лабе 5 добавим обязательный зелёный CI.

Проверка — push в main должен быть отклонён:

```
git commit --allow-empty -m "test: direct push"
git push                    # rejected ... GH013: Repository rule violations found
git reset --hard HEAD~1     # убрать тестовый пустой коммит локально
```

4. Задачи и доска

Создайте 3 issue:

1. `docs: README с инструкцией запуска`
2. `chore: шаблон Pull Request и Dependabot`
3. `feat: переименование проекта`

Projects → *New project* → *Board*. На доске: + *Add item* → введите `#` → выберите issue. Так добавьте все три.

Закрытые issue доска сама переносит в **Done** (правило *Workflows* → *Item closed*, включено по умолчанию).

5. README через PR с ревью

```
git switch -c docs/1-readme
```

Создайте `README.md`:

- что за сервис и его основные эндпоинты;
- как запустить: `.env`, `docker compose up -d --build --wait`;
- как разрабатывать: `docker compose up -d --wait db` + F5 в Visual Studio;
- как запустить тесты.

```
git add README.md
git commit -m "docs: add README with run instructions"
git push -u origin docs/1-readme
```

На GitHub: *Compare & pull request* (если кнопки уже нет — *Pull requests* → *New pull request*), в описании `Closes #1`, ревьюер — **B**. **B:** оставляет минимум один комментарий к строке. A исправляет новым коммитом, B делает *Approve*. **A:** *Squash and merge*. Убедитесь, что issue #1 закрылся и уехал в **Done**.

6. Конфликт (один раз, в репозитории A)

Оба начинают от свежего `main` — A в своей папке, B в `C:\dev\demo-api-<A>`:

```
git switch main; git pull
git switch -c docs/title-a        # у B: docs/title-b
```

Оба меняют **одну и ту же строку** — первую строку `README.md` с заголовком — по-разному. Коммит, push, PR.

Первый PR мержится. Во втором GitHub пишет *This branch has conflicts*. Разрешает автор второго PR у себя:

```
git fetch origin
git merge origin/main             # CONFLICT
# откройте README.md (Visual Studio покажет редактор слияния), оставьте правильный вариант
git add README.md
git commit -m "docs: resolve title conflict"
git push
```

Смержите второй PR.